Actas de las XX JENUI. Oviedo, 9-11 de julio 2014
ISBN: 978-84-697-0774-6
Páginas: 237-243

Uso de JUnit para evaluación en laboratorio de Estructuras de Datos

Agustín Cernuda del Río, Miguel García Rodríguez,
Néstor García Fernández, Martín González Rodríguez
Departamento de Informática
Universidad de Oviedo
Asturias
{guti, garciamiguel, nestor, martin}@uniovi.es

Resumen

Las pruebas del software, y en especial las pruebas unitarias automatizadas, son un área relevante de las competencias de un titulado en Informática. Pero aparte de su papel en los contenidos del currículo, pueden ser una herramienta útil para la docencia.

Se ha documentado en múltiples ocasiones el uso de las pruebas automatizadas para autoevaluación de los estudiantes, o para implementar sistemas de corrección automáticos o semiautomáticos; es decir, en diversas formas de trabajo autónomo. Sin embargo, la aportación de este artículo radica en documentar la aplicación de una variante, que consiste en utilizar pruebas automatizadas durante los propios exámenes prácticos, en una doble vertiente: como instrumento del profesor para hacer posible una evaluación rápida, ordenada y coherente en el momento, y simultáneamente como elemento integrante del propio ejercicio.

Cabe anticipar algunos potenciales problemas del uso de esta técnica. Aquí se discuten dichos problemas y su incidencia real tras la experiencia.

Abstract

Software testing, and automated unit testing in particular, is a relevant topic for software engineering studies. But, in addition to their role as a part of the curricula, they are a tool that can be applied in teaching itself.

The use of automated testing in teaching, be it for student self-assessment, or for implementing automatic or semiautomatic grading systems, is well documented. These approaches are largely based on self-study or autonomous work schemas. However, this paper presents an approach based on applying automated software testing during lab exams, with a double goal: on one hand, as a tool for the teacher, that makes it possible to perform a quick, orderly and consistent assessment in-place, and on the other hand, as an important component of the exam itself.

Some potential problems can be identified in advance. In this paper, these problems are discussed and their real relevance in practice is evaluated.

Palabras clave

Evaluación, pruebas unitarias, JUnit, laboratorio, programación.

1. Motivación

La progresiva madurez de la industria del desarrollo de software ha traído consigo una mayor presencia de las pruebas automatizadas en los procesos de desarrollo. Del mismo modo, en la enseñanza de titulaciones informáticas las pruebas automatizadas de software han ido encontrando diversos campos de aplicación, que podemos organizar en dos grandes campos:

- Las pruebas son el propio objetivo formativo. Se han adoptado interesantes iniciativas destinadas a mejorar la destreza en la elaboración de las pruebas [10], a percibir y comprender su utilidad en el proceso [8] o a cambiar el planteamiento del desarrollo hacia un modelo dirigido por pruebas [3].
- Las pruebas son un instrumento que, aparte de transmitir algunas de las ideas anteriores, persigue objetivos adicionales o incluso prioritarios, como facilitar la evaluación en entornos masificados [2, 4], promover la abstracción y el bajo acoplamiento [7], favorecer la autoevaluación de los alumnos, como infraestructura de trabajo en prácticas organizadas en torno a la simulación [5], etc.

El presente artículo describe una experiencia que se enmarca en el segundo grupo. Se pretende que el alumno perciba el carácter habitual de las pruebas

unitarias en la construcción de software, pero también facilitar la logística de actividades de evaluación realizadas en clase, tanto en lo referente a la gestión del tiempo como en la aplicación de criterios objetivos y no ambiguos a la hora de calificar los trabajos.

No obstante, no se desea en modo alguno abordar la implantación de una plataforma concreta de trabajo destinada a realizar una evaluación plenamente automatizada. Se pretende utilizar las pruebas unitarias de manera plenamente flexible y bajo un enfoque ligero, sin la necesidad de preparar infraestructura alguna y sin coste de adaptación más allá del trabajo normal de preparación docente.

Además, un componente importante de este trabajo es la aplicación de las pruebas de manera presencial, en el momento; muchos otros trabajos citados aquí [2, 4, 5, 6, 7] abordan el uso de las pruebas en el trabajo autónomo del alumno o del profesor, pero los autores tenían ciertas dudas sobre la utilidad de la herramienta para un uso inmediato y en tiempo real.

2. Antecedentes

2.1. Entorno académico

La experiencia descrita tuvo lugar en la asignatura Estructuras de Datos, asignatura de primer semestre del segundo curso del Grado en Ingeniería Informática del Software, impartida en la Escuela de Ingeniería Informática de la Universidad de Oviedo, que engloba a unos 140 alumnos. Incluye clases teóricas, seminarios y clases prácticas de laboratorio, desarrolladas en Java con la herramienta Eclipse; es en estas últimas donde se ha realizado la experiencia.

En los laboratorios (grupos de 10 a 15 alumnos) los alumnos desarrollan sucesivos mini-proyectos de implementación de estructuras de datos y experimentos asociados, de manera que realicen un cierto aprendizaje constructivo, por descubrimiento. Podrían calificarse de prácticas dirigidas, en el sentido de que en cada sesión (de dos horas) hay unos objetivos de implementación concretos, que el alumno normalmente completa con su trabajo autónomo antes de la siguiente sesión.

Así pues, todos los alumnos trabajan aproximadamente a un mismo ritmo, sobre los mismos problemas y aspectos de implementación, pero no son prácticas completamente cerradas. Se facilitan unas especificaciones de interfaces bastante concretas, pero no exactas en todos sus extremos [7]. Siempre se dejan pequeños márgenes de variación o se detallan distintas alternativas que los alumnos pueden elegir. Esto tiene el beneficio de que el alumno practique también este tipo de toma de decisiones.

Los laboratorios se evalúan mediante evaluación continua, controlando el trabajo en cada sesión. A modo de hitos, para evitar plagios y para comprobar la adquisición de las competencias necesarias, se establecen *sesiones de evaluación* en fechas concretas. Durante el semestre, se han realizado tres de estas sesiones, según se detalla en el Cuadro 1.

Unidad didáctica	**Evaluación**
1- Fundamentos de algoritmia, complejidad	Sesión 1
2- Grafos	
3- Árboles y montículos	Sesión 2
4- Tablas hash	Sesión 3

Cuadro 1: Unidades didácticas y sesiones específicas de evaluación en 2013.

Contenido	**Ejercicio**
Grafo	- Longitud del camino de longitud mínima (a partir de Floyd y dados otros elementos básicos necesarios)
Árbol AVL	- Obtener número de nodos - Ver si es completo - Hallar altura - Comprobar si es AVL (dado cálculo de altura)
Tabla hash	- Calcular tasa de colisiones

Cuadro 2: Algunos ejemplos de ejercicios de evaluación. En el propio enunciado se ofrecen pistas si la implementación no es fácilmente comprensible, y se suministra código adicional si la tarea puede ser larga.

En ellas se pide alguna pequeña modificación o ampliación de lo que se haya implementado hasta entonces. El tiempo disponible suele ser de veinte minutos; son modificaciones relativamente sencillas, que requieren simplemente una comprensión clara de los propios programas y del funcionamiento de las estructuras de datos. Normalmente basta con cosas tales como añadir contadores, combinar funciones ya existentes o implementar sencillos algoritmos. Caracterizar en detalle ejercicios y contenidos excedería la extensión recomendable para este artículo, pero a modo de orientación se ofrecen algunos ejemplos reales en el Cuadro 2.

Pasados los veinte minutos, el profesor evalúa en el momento, con cada uno de los alumnos, el trabajo realizado.

2.2. Dificultades prácticas

Puede ser asumible realizar actividades de evaluación durante las horas de clase, pero la corrección inmediata del trabajo plantea más problemas.

En años anteriores, se pedía a los alumnos imprimir por pantalla el estado de las estructuras de datos, y el profesor podía examinar esos resultados. Pero esta era

una tarea trabajosa y compleja; no permitía un análisis pausado y no resultaba fácil deducir calificaciones. Además, había una cierta percepción de "todo o nada", en la que se evaluaba el resultado final de los algoritmos. Por otra parte, la casuística hacía difícil obtener calificaciones concretas y comparables, puesto que además el criterio iba conformándose a medida que el profesor veía unos u otros fallos.

2.3. Posible uso de pruebas automatizadas

La automatización de las pruebas puede parecer una solución obvia. El enfoque sería el siguiente:

- El alumno hace el ejercicio.
- Termina el tiempo del examen.
- El profesor toma sus casos de prueba y los ejecuta sobre el ejercicio del alumno.

Esto en principio acarrea también algunos inconvenientes. Para aplicar las pruebas directamente, el código del alumno tendría que ofrecer interfaces perfectamente compatibles con las pruebas. Esto es factible bajo ciertos enfoques [7], pero en esta asignatura se persigue un equilibrio entre los ejercicios dirigidos y una cierta aportación del alumno. Si el alumno recibe directrices completas y cerradas, trabajará solamente contra las especificaciones, olvidando el enfoque constructivo de la solución.

Aun con interfaces definidas y documentadas, rara vez las pruebas encajarán de forma inmediata, salvo que los profesores sean exhaustivos en su especificación y los alumnos la interpreten sin errores.

Por todo lo anterior, se decide abordar un esquema distinto de aplicación que se describe en el apartado siguiente.

3. Planteamiento de la experiencia

A continuación se describe cómo se ha decidido utilizar JUnit en la práctica para evitar los inconvenientes descritos. Básicamente, la idea consiste en que en vez de utilizar las baterías de pruebas estrictamente como un medio de verificación a posteriori, se incorporen estas pruebas al propio ejercicio que el alumno realiza.

3.1. Procedimiento propuesto

- Durante las sesiones habituales de laboratorio, los alumnos son instruidos para realizar sus propias pruebas. El profesor también suele suministrar pruebas básicas.
- En la sesión de evaluación, los alumnos reciben en primer lugar el enunciado del problema (en papel) y las explicaciones oportunas. Este enunciado describe qué ficheros deben descargar a su ordenador (tanto código adicional como ficheros de datos), incluyendo las baterías de pruebas que el profesor utilizará para evaluar (más completas que las básicas ofrecidas previamente).
- En cuanto se han descargado los ficheros, los equipos son desconectados de la red y el tiempo empieza a contar.
- Los alumnos adaptan sus programas para que encajen con las pruebas. Estas pruebas, además de probar la nueva funcionalidad propuesta, verifican el funcionamiento de lo anterior, de forma más completa que las suministradas previamente de manera informal.
- La adaptación suele requerir ajustes de nomenclatura o signatura, algún cambio de visibilidad de método, o implementar algún método de acceso. También alguna pequeña modificación en la gestión de errores. En algunos casos, los alumnos comentan líneas de código de las pruebas que son incompatibles con su implementación, aunque en ese caso están obligados a notificarlo durante la revisión.
- En los veinte minutos de desarrollo de las pruebas los alumnos realizan tanto la adaptación necesaria como la implementación de la nueva funcionalidad solicitada.
- Terminado el tiempo, son conectados a la red y entregan sus ejercicios en el campus virtual.
- Entonces abandonan momentáneamente la sala, y el profesor los va llamando para revisar sus ejercicios. La evaluación consiste en ejecutar las pruebas y, en algún caso, examinar en qué punto un bloque de pruebas falla o qué líneas el alumno ha tenido que comentar.

Este enfoque se ha aplicado por primera vez durante el presente curso. Cabía anticipar ciertas ventajas e inconvenientes.

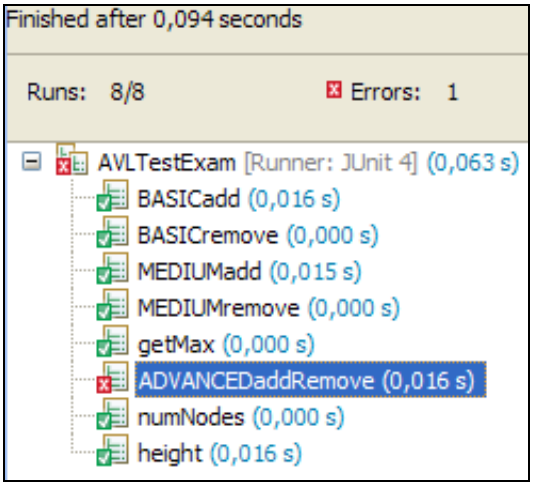

Figura 1: Ejecución de un banco de pruebas de evaluación.

3.2. Problemas potenciales

Al igual que en el apartado 2.3 se expuso el procedimiento básico y también los problemas potenciales, en este caso también cabe prever ciertos riesgos, entre los que se encuentran los siguientes.

- Que la preparación de estas pruebas representase un esfuerzo significativo del profesor.
- Que los alumnos empleasen demasiado tiempo en adaptar su código a las pruebas, solventando discrepancias formales. Las pruebas se convertirían así en una interferencia indeseada durante la sesión de evaluación.
- En línea con lo anterior, que la sesión de evaluación resultase demasiado larga y no fuese viable.
- Que los alumnos acabasen centrándose en escribir código para pasar las eventuales pruebas, y no en construir estructuras de datos.
- Que resultase difícil verificar algunos aspectos del trabajo de los alumnos. No es posible realizar pruebas de caja blanca como tales (recuérdese que las prácticas son dirigidas, pero no totalmente cerradas).

3.3. Ventajas esperadas

Las pruebas básicas suministradas en las sesiones ordinarias sirven como autodiagnóstico para los alumnos que no realizan el trabajo semanal o tienen más dificultades para llegar a los mínimos, que pueden conocer pronto su situación y sus posibilidades. Por otro lado, sirven como reto para los alumnos de un perfil alto, que tratan de predecir las variaciones que el profesor va a realizar sobre esas pruebas, haciendo que prueben de una forma mucho más profunda y efectiva su código.

Las pruebas para la sesión de evaluación están organizadas en bloques significativos (aspectos básicos, avanzados...) También se pueden escribir pruebas específicas para cada aspecto de la materia implicada (casos más o menos complejos de inserción y borrado, por ejemplo). Aplicando un criterio aproximado a los resultados de JUnit (puntuando por bloques y estableciendo mínimos) la calificación es relativamente objetiva y fácil de explicar al alumno.

Respecto al alumno, como en todo uso de pruebas automatizadas es el propio programador el que obtiene realimentación inmediata. Antes de que terminen los 20 minutos, el alumno ya sabe en líneas generales cómo le ha ido.

Cuando alguna prueba no pasa de forma satisfactoria, la interfaz de usuario de JUnit (en este caso bajo Eclipse) facilita por su propia naturaleza el acudir inmediatamente al código implicado, y el profesor sabe inmediatamente, con una alta probabilidad, qué tipo de problema tiene el código del alumno.

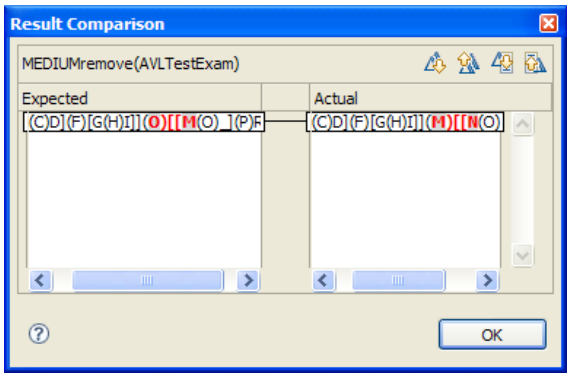

Figura 2: Identificando discrepancias en un árbol AVL.

Otra ventaja es que, aunque el criterio de evaluación es análogo para todos los grupos, cada profesor revisa y/o escribe sus propios bancos de pruebas, teniendo en cuenta las particularidades o pequeñas diferencias de enfoque entre un grupo y otro.

4. Desarrollo de la experiencia

Al llevar a la práctica el planteamiento descrito, se ha podido comprobar el grado de viabilidad de este sistema, sus ventajas y sus inconvenientes en la práctica.

4.1. Aplicación en la práctica

Este sistema se ha aplicado para evaluar a unos 140 alumnos, siempre dentro de las horas de clase y sin exceder el tiempo asignado a cada sesión.

Respecto a las ventajas, el resultado ha sido conforme a lo esperado. La organización de las pruebas en bloques de la granularidad deseada facilita el examen; un rápido vistazo a la ventana de resultados de JUnit (Figura 1) permite saber qué bloques han pasado, y esto es casi un adelanto de la calificación. Además, es fácil aislar casos de prueba especialmente elegidos, que permiten discriminar los trabajos más completos y brillantes. Por supuesto, hay una parte de la calificación que no emana "ciegamente" de los resultados, sino del criterio del profesor, y se basa en la calidad del código, la documentación del mismo, los casos de prueba desarrollados por el alumno, su actitud en clase, etc. Esta parte de la calificación puede completarse posteriormente, pero el alumno obtiene realimentación rápida y detallada sobre la calidad funcional de su trabajo. En lo que respecta a la revisión en el propio laboratorio después de esos 20 minutos, para el profesor es una sencilla toma de datos, pero para el alumno es más bien un ejercicio de recapitulación, porque no hay novedades o sorpresas. El alumno no sale del laboratorio con la calificación definitiva (como se ha dicho, hay una parte de la misma que procede de la reflexión del profesor), pero sí con una aproximación muy razonable.

La experiencia ha demostrado que, efectivamente, usando la interfaz de usuario de JUnit resulta fácil identificar el tipo de error cometido; el profesor no se ve implicado en una sesión de depuración como venía ocurriendo a veces de manera indeseada, sino realmente en una sesión de evaluación.

Respecto a la posibilidad de adaptar las pruebas a cada grupo, este sistema se ha aplicado en un grado bilingüe, con cuatro profesores implicados y grupos en inglés y en español; más allá de traducir los enunciados, no ha habido problemas de coordinación o logística. Compartiendo con antelación los bancos de pruebas, cada profesor tenía una idea clara de los criterios de evaluación que iban a aplicar sus compañeros, y los adaptaba en caso necesario.

Discutiendo los problemas potenciales, para los profesores tampoco ha resultado un sobreesfuerzo la redacción de pruebas unitarias respecto a la preparación del examen. Hay que tener en cuenta que en general los profesores de la asignatura implementan los ejercicios que luego piden a sus alumnos, y esto implica también realizar las correspondientes pruebas unitarias de todos modos. Preparar las pruebas para los alumnos implica en general realizar selecciones o reorganizaciones de este trabajo ya realizado.

El problema mencionado respecto a las pruebas de caja blanca (apdo. 3.2) se soslaya en gran medida suministrando a los alumnos funciones de volcado de la estructura de datos (versiones del método **`toString()`** o similares). Si se tiene una representación conocida del estado de la estructura de datos, el profesor puede escribir pruebas automatizadas sobre dicha representación y comprobar en todo momento cómo evoluciona. Esto permite identificar fallos muy concretos en los algoritmos, o incluso señalar exactamente qué se ha hecho mal, simplemente viendo la discrepancia entre el volcado obtenido y el esperado, cosa que JUnit facilita (Figura 2).

Por lo que se refiere a la interferencia de las pruebas en la verdadera evaluación, el interés de este enfoque reside en que la adaptación a las pruebas se incorpora de manera efectiva al ejercicio, forma parte de él. Al fin y al cabo, se persigue comprobar que el alumno está familiarizado con su código y es capaz de resolver adaptaciones y problemas sencillos con él; el hacer que estas pruebas funcionen puede parecer una interferencia antes del "ejercicio real", pero en el caso que se presenta, se decidió que fuesen precisamente parte del "ejercicio real".

Los alumnos han reaccionado a este sistema de manera apreciable durante el curso. La primera actividad de evaluación tuvo un efecto sorpresa y arrojó pobres resultados. Varios alumnos relativamente activos o brillantes obtuvieron bajas calificaciones, a pesar de que se había descrito claramente la mecánica y se habían suministrado las pruebas básicas, advirtiendo de que eran sólo un punto de partida.

En muchos casos, la principal dificultad del ejercicio no residió en el propio ejercicio, ni en adaptar el código a las pruebas, sino a problemas en el trabajo precedente, que se manifestaron durante la evaluación. En ese sentido, es positivo que los alumnos no se jugaron el resultado en veinte minutos de inspiración, sino que los principales problemas venían del trabajo previo. Sus pruebas unitarias habían adolecido de los problemas que la literatura siempre señala: escribían pruebas muy livianas, más destinadas a reforzar su idea de que el trabajo era correcto que a comprobar realmente si lo era o no. Cuando los profesores suministraron sus pruebas básicas, a pesar de todas las advertencias al respecto los alumnos consideraron que era más que suficiente con pasar estas. Las pruebas de la evaluación, más completas, pusieron de relieve las deficiencias del trabajo realizado. Deficiencias que, eso sí, si son graves es muy difícil corregir en esos veinte minutos (no es el momento, y no es el objetivo de la sesión de evaluación).

Esa primera evaluación tuvo un efecto notable. En el siguiente ejercicio, muchos alumnos estaban "en guardia" e intentaron hacer pruebas más profundas. Los alumnos que respondían a este perfil obtuvieron calificaciones sensiblemente más altas en las siguientes pruebas. También se percibió un notable incremento en la cantidad y calidad de sus pruebas.

No obstante, hay que hacer notar que lo anterior se refiere a alumnos de un cierto perfil de calificación y participación, y en todo caso resulta muy difícil extraer conclusiones generalizables. A fin de evaluar plenamente la experiencia, los autores se plantearon realizar diversas mediciones y evaluaron los datos, y se podría haber incluido en este artículo una selección de información que parecería confirmar la bondad del método; pero pronto se vio que los mismos problemas que afectan en general a las métricas del software harían discutible este enfoque, combinados además con la especial dificultad inherente a las mediciones en el campo de la pedagogía. Por ejemplo, se realizó un análisis preliminar del código de pruebas de todos los alumnos, contando líneas de código para comprobar si había alguna pauta de evolución; pero aparte de ciertas dificultades prácticas (habría que descontar el código de pruebas suministrado por los profesores, o discernir qué ficheros del alumno están realmente activos) había otras de fondo (medir la calidad de unas pruebas por el número de líneas de código es una aberración, y además las cifras obtenidas no son comparables entre un ejercicio y otro, ya que el simple código de inicialización de un grafo o árbol puede tener un tamaño que haga parecer irrelevante el verdadero código de pruebas). En todo caso, la experiencia no se llevó a cabo en condiciones que permitieran achacar las mejorías al método, ya que hubo muchos otros factores. En consecuencia, se abandonó la pretensión de presentar aquí los datos, porque de

todos modos no permitían obtener conclusiones fiables. Lo mismo puede decirse, como corolario de lo anterior, respecto a correlaciones entre el código de pruebas y las calificaciones finales.

Un efecto indeseado de esta técnica, que se había identificado como problema potencial y se materializó en cierto modo, es que se produjo un cierto enfoque finalista de las pruebas. A pesar de la insistencia en que los alumnos fueran redactando pruebas para todas las etapas parciales y componentes de su trabajo a medida que lo construían, sus pruebas solían seguir centradas en el funcionamiento global de la estructura de datos.

5. Conclusiones

Las conclusiones de esta experiencia pueden resumirse como sigue.

Además del uso cada vez más extendido de JUnit como elemento de trabajo autónomo o a posteriori, es posible utilizar JUnit como medio para realizar evaluaciones de trabajos prácticos de programación en clase. En el caso aquí descrito, se trata de prácticas dirigidas, pero no completamente especificadas, y la evaluación es continua, pero en la sesión de evaluación se pretende simplemente comprobar que el alumno está familiarizado con su código y comprende el funcionamiento de sus estructuras de datos, siendo capaz de realizar pequeñas modificaciones.

Bajo estas premisas, la adaptación del código a los bancos de pruebas se incorpora como parte del ejercicio de evaluación, de forma natural y no ajena a sus objetivos.

De este modo, es posible realizar en un corto espacio de tiempo evaluaciones objetivas y completas, sin necesidad de recurrir a plataformas complejas ni instalar sistema alguno. Entre las pruebas de caja negra y el uso de volcados de los datos para sustituir las pruebas de caja blanca, se puede caracterizar de forma muy estrecha el trabajo que los alumnos han realizado.

Algunos de los potenciales problemas con la duración del examen se solventan suministrando al alumno el código adicional que necesite (en especial funciones de volcado de datos), de modo que este sólo tenga que realizar pequeñas adaptaciones. La experiencia demuestra que estas adaptaciones no interfieren gravemente en el desarrollo del ejercicio ni suscitan quejas. Esto es así a pesar de que los ejercicios de los alumnos presentan ligeras variaciones entre sí.

El uso habitual de las pruebas como parte de la evaluación contribuye a identificarlas como algo inherente al trabajo de programación. Por otra parte, los alumnos perciben las pruebas propias como una preparación para asegurar el éxito en el examen, papel que se asemeja al que desempeñan las pruebas en la producción real de software en relación con los clientes finales.

La granularidad de los bancos de pruebas permite al profesor organizar dichas pruebas en bloques muy específicos, anulando el peligro del "todo o nada" en la calificación. Es posible identificar de forma bastante precisa las deficiencias en el trabajo del alumno, y también qué trabajos son especialmente sólidos o completos.

Un posible efecto negativo es que los alumnos, en lugar de ir haciendo pruebas unitarias para todas sus funcionalidades parciales como se les solicita, se basan demasiado en las pruebas básicas que los profesores van suministrando en clase (que suelen comprobar el funcionamiento final de las clases). En este sentido, siguen haciendo una interpretación finalista de las pruebas, como se desprende de su aplicación en el examen. Está por ver cómo se puede remediar esto para que perciban las pruebas como un instrumento que deben aplicar también en sus propias etapas intermedias, componente a componente.

Referencias

[1] Enrique A. de la Cal, Marco A. García y María José Suárez-Cabal. Sistemas de Computación 2003-2008: corrección de caja negra vs "pruebas de aceptación de usuario". En *Actas de las XV Jornadas de Enseñanza Universitaria de Informática, Jenui 2009*, pp. 169 – 176, Barcelona, julio 2009.

[2] C. Douce, D. Livingstone, J. Orwell, S. Grindle y J S Cobb. A Technical Perspective on ASAP – Automated System for Assessment of Programming. En *9th International Conference on Computer Aided Assessment*, julio 2005, Loughborough, Reino Unido.

[3] Stephen H. Edwards. Rethinking Computer Science Education from a Test-first Perspective. *OOPSLA '03*, octubre 2003, Anaheim, California, EEUU.

[4] Antonio García Dopico, Santiago Rodríguez de la Fuente y Francisco Javier Rosales García. Automatización de prácticas en entornos masificados. En *Actas de las IX Jornadas de Enseñanza Universitaria de Informática, Jenui 2003*, pp. 119 – 126, Cádiz, julio 2003.

[5] P. Pablo Garrido Abenza. Sistema de evaluación automatizada de prácticas para Tecnología de Computadores. En *Actas de las XI Jornadas de Enseñanza Universitaria de Informática, Jenui 2005*, pp. 138 – 144, Villaviciosa de Odón (Madrid), julio 2005.

[6] Piedad Garrido Picazo, Gabriel Fuertes Muñoz y Jesús Tramullas Saz. Una herramienta para la evaluación automatizada de la disciplina de Bases de Datos. En *Actas de las XIII Jornadas*

de Enseñanza Universitaria de Informática, Jenui 2007, pp. 547 – 548, Teruel, julio 2007. Póster.

[7] Michael T. Helmick. Interface-based Programming Assignments and Automatic Grading of Java Programs. *ITiCSE'07*, junio 2007, Dundee, Escocia, Reino Unido.

[8] Lucas Layman, Travis Cornwell y Laurie Williams. Personality Types, Learning Styles, and an Agile Approach to Software Engineering Education. SIGCSE'06, marzo 2006, Houston, Texas, EEUU.

[9] Carlos López, Raúl Marticorena y David H. Martín. Pruebas de caja negra: una experiencia real en laboratorio. En *Actas de las XI Jornadas de Enseñanza Universitaria de Informática, Jenui 2005*, pp. 189 – 196, Villaviciosa de Odón (Madrid), julio 2005.

[10] Macario Polo Usaola y Pedro Reales Mateo. Enseñanza de la mutación en pruebas de software. En *Actas de las XVIII Jornadas de Enseñanza Universitaria de Informática, Jenui 2002*, pp. 1 – 8, Ciudad Real, julio 2012.